Escuela Politécnica Superior

Reconocimiento automático de símbolos kanji

Grado en Ingeniería en Sonido e Imagen en Telecomunicación

Trabajo Fin de Grado

Autor:
Jonatan Puche Andujar

Tutor/es:
Francisco Antonio Pujol López

Julio 2020

Universitat d'Alacant
Universidad de Alicante

Resumen

Este proyecto se centra en el diseño y creación de una aplicación de estudio móvil capaz de reconocer símbolos kanji por medio de un dibujo y así ayudar a personas de nivel principiante con la lectura de uno de los alfabetos más complicados. Para lograr este fin, se ha decidido implementar una red neuronal que sea capaz de aprender a reconocer los símbolos kanji de grado 1, recogidos en la lista jōyō kanji, de primer año de primaria. Son un total de 80 kanjis. El dataset será personalizado para esos 80 kanjis y serán obtenidos de los dibujos realizados en la propia aplicación con un total de 500 muestras por kanji. Se definirá la vista general de la aplicación, así como el flujo de trabajo dentro de la misma para que la interacción usuario/aplicación sea lo más fluida posible. Se compararán dos modelos diferentes de redes neuronales, un clasificador multicapa y una red neuronal convolucional. Se estudiarán técnicas y métodos eficientes para la creación del dataset, para que la fase de entrenamiento sea lo más coherente posible. Posterior a la fase de entrenamiento se analizarán los resultados obtenidos y se compararán con proyectos similares en forma. El presente proyecto es capaz de reconocer figuras kanji sin necesidad de conocer la forma de dibujarlos en el orden correcto, como es tradición en Japón. Los modelos propuestos han sido validados experimentalmente obteniendo unos resultados en precisión de entorno a un 98%, lo que indica que la herramienta propuesta es más que válida para ayudar con la etapa de aprendizaje de una persona principiante en el idioma japonés.

Contenido

Índice de figuras

1. Introducción

1.1. Motivación y justificación

El japonés figura actualmente en la lista de los idiomas más aprendidos a nivel mundial, quizá por su exotismo o simplemente por curiosidad, cada vez es más la gente que se anima a aprenderlo.

El aprendizaje de un idioma extranjero no es una tarea sencilla, sobre todo, si no comparte una base común como puede ser el alfabeto, la similitud de vocabulario, la fonética o la composición léxica de una frase. En el caso del japonés, ninguna de las anteriores características se aplica, a excepción de la fonética, que es notablemente similar a la del castellano. La construcción de la frase es totalmente diferente, el vocabulario no guarda relación con la lengua latina e incluso el alfabeto es diferente, ya que, el japonés cuenta con tres tipos de sistemas de escritura diferentes.

El primero de ellos se denomina *hiragana* y es originario de Japón. En él, se representan las vocales y sílabas que dan forma a palabras. El segundo es un derivado del *hiragana* conocido como *katakana*. Si bien representa las mismas sílabas que el *hiragana* la simbología cambia casi por completo. Es usado para hacer referencia a palabras de origen extranjero que no existen en el vocabulario japonés. Ambos son sistemas de escritura silábicos y cuentan con un total de 46 caracteres cada uno.

El último sistema de escritura en cambio es ideográfico y se denomina *kanji*. Esto quiere decir que se emplean símbolos para representar una idea. El sistema de escritura *kanji* es heredado de la escritura china y sigue vigente en la actualidad, aunque si bien simplificado, con un uso extenso en el día a día de Japón. Es indispensable saber interpretar cada uno de los *kanjis* y más importante aún, saber que lectura otorgar a cada símbolo dependiendo del contexto en el que sea escrito. La gran mayoría de símbolos *kanji* cuenta con las lecturas china y japonesa y se puede dar el caso de que el símbolo pueda tener múltiples lecturas en ambas.

Los *kanjis* cuentan con 2136 *kanji* de uso común considerados por el Ministerio de educación japonés y más de 11000 *kanji* incluidos en cualquier sistema de computación actual.

Uno de los principales métodos de estudio se basa en la lectura del idioma. Extraer vocabulario de frases es fundamental para sumergirte en la cultura de la región y ayuda a comprender la formación de las mismas y la expresión empleada. Inevitablemente se deben aprender tanto *hiragana* como *katakana,* así como también los numerosos *kanjis* que se emplean en la escritura japonesa del día a día, lo que provoca que la lectura aumente en dificultad de forma considerable.

Teniendo en cuenta que en una misma frase aparecen mezclados los tres sistemas de escritura mencionados, se requiere la identificación correcta de los símbolos que van apareciendo para traducir la frase y apuntar nuevo vocabulario. Sin embargo, los *kanjis* tienen la complejidad adicional de que pueden ser representados con múltiples sílabas dentro de diferentes contextos en las frases. El mayor problema surge cuando no sabes de que *kanji* se trata, siendo imposible continuar con la lectura de la frase, provocando muchas veces frustración en las personas con un nivel principiante, ya que no tienes otra opción que buscar en todos los 2136 *kanjis* hasta que des con el indicado.

1.2. Objetivos

1.2.1. General

La propuesta principal de este trabajo consiste en facilitar la fase de aprendizaje de personas que inician su aventura en el idioma japonés, dotándoles de una herramienta que complemente la lectura, con una aplicación de móvil que permita dibujar el símbolo *kanji* que el lector tenga duda porque no recuerda cómo se pronuncia o simplemente porque nunca lo había visto antes. La aplicación ofrece a su vez la gradual memorización de los caracteres buscados, aumentando la velocidad de aprendizaje de la persona, siendo en última instancia el objetivo global a conseguir.

El sistema de *kanji* en Japón se enseña por niveles de estudio. La aplicación a desarrollar contempla los *kanjis* enseñados en primaria incluidos en los primeros años de primaria con un total de 80 caracteres *kanji*, y mediante reconocimiento de imagen y aprendizaje de máquina se pretende dar una solución satisfactoria y útil a la hora de buscar el *kanji* para su estudio.

1.2.2. Específicos

1. Crear la interfaz de usuario multiplataforma
2. Construir una base de datos kanji para entrenar una red neuronal
3. Comprender los diferentes tipos de redes neuronales que existen
4. Procesar las imágenes para crear consistencia en el conjunto de datos de los diversos kanjis
5. Comparar diversas técnicas de creación de redes neuronales
6. Diseñar dos modelos con técnicas de redes neuronales diferentes y comparar resultados

2. Marco teórico

2.1. Machine Learning

El Aprendizaje Automático es una disciplina de las ciencias informáticas, relacionada con el desarrollo de la Inteligencia Artificial, y se utiliza para crear sistemas con la capacidad de aprender patrones en estructuras de datos complejas.

Se resume mejor con esta declaración frecuentemente citada, hecha por Arthur Samuel en 1959:

> *"[Aprendizaje de máquina es el] campo de estudio que da a las computadoras la habilidad de aprender sin ser programadas explícitamente."* [1]

Es una tecnología que permite automatizar una serie de operaciones con el fin de agilizar un proceso complejo y costoso. Supone una gran ventaja a la hora de controlar una ingente cantidad de información de un modo mucho más efectivo.

La máquina no aprende por sí misma, sino por medio de un algoritmo, que se va modificando con la constante entrada de datos y que puede predecir escenarios futuros o tomar acciones de manera automática según ciertas condiciones.

2.1.1. Aprendizaje supervisado

Es una de las estrategias más usadas en el campo de aprendizaje de máquina. Se emplea el término supervisado porque los algoritmos que aprenden de un conjunto de datos (dataset) disponen de etiquetas que guían el proceso de aprendizaje. [2]

Mediante un proceso iterativo, comparando el resultado del algoritmo con lo que se espera obtener, se minimiza el error producido por la iteración anterior hasta llegar a una solución que sea aceptable.

Una vez que se le ha proporcionado la suficiente cantidad de datos, podrán introducirse nuevos sin necesidad de etiquetar para realizar predicciones basadas en la fase de entrenamiento.

Algunos ejemplos populares de algoritmos de aprendizaje automático supervisados son:

- Regresión lineal para problemas de regresión.
- Bosque aleatorio para problemas de clasificación y regresión.
- Máquinas de soporte vectorial para problemas de clasificación.

2.1.2. Aprendizaje no supervisado

El aprendizaje no supervisado interviene cuando tienes datos de entrada, pero la relación entrada/salida no está definida como en el aprendizaje supervisado. [3]

En este tipo de aprendizaje no se usan valores verdaderos o etiquetas. Estos sistemas tienen como finalidad la comprensión y abstracción de patrones de información de manera directa. Es un modelo que se conoce como clustering, método de entrenamiento similar al modo en que los humanos procesan la información.

Todo el proceso de modelado se lleva a cabo sobre un conjunto de ejemplos formado tan sólo por entradas al sistema. No se tiene información sobre las categorías de esos ejemplos. Por lo tanto, en este caso, el sistema tiene que ser capaz de reconocer patrones para poder etiquetar las nuevas entradas.

Algunos ejemplos populares de algoritmos de aprendizaje no supervisados son:

- K-clustering para problemas de agrupación en clústeres.
- Algoritmo Apriori para problemas de aprendizaje de reglas de asociación.

2.1.3. Aprendizaje semi-supervisado

En ciencias de la computación, el aprendizaje semi-supervisado es una clase de técnicas de aprendizaje automático que utiliza datos de entrenamiento tanto etiquetados como no etiquetados: normalmente una pequeña cantidad de datos etiquetados junto a una gran cantidad de datos no etiquetados. El aprendizaje semi-supervisado se encuentra entre el aprendizaje no supervisado (sin datos de entrenamiento etiquetados) y el aprendizaje supervisado (con todos los datos de entrenamiento etiquetados). Los investigadores del campo del aprendizaje automático han descubierto que los datos no

etiquetados, cuando se utilizan junto a una pequeña cantidad de datos etiquetados, pueden mejorar de forma considerable la exactitud del aprendizaje. La adquisición de datos etiquetados para resolver un problema suele requerir un agente humano capacitado para clasificar de forma manual los ejemplos de entrenamiento. El coste asociado al proceso de etiquetado puede hacer que un conjunto de entrenamiento totalmente etiquetado sea inviable, mientras que la adquisición de datos sin etiquetar es relativamente poco costosa. En estos casos, el aprendizaje semi-supervisado puede ser muy útil. [4]

2.1.4. Aprendizaje por refuerzo

En la técnica de aprendizaje mediante refuerzo, los sistemas aprenden a partir de la experiencia. Como ejemplo se puede observar el comportamiento de un coche autónomo. Cuando el vehículo toma una decisión errónea, es penalizado, dentro de un sistema de registro de valores. Mediante dicho sistema de premios y castigos, el vehículo desarrolla una forma más efectiva de realizar sus tareas.

El aprendizaje de refuerzo (RL) es un área de aprendizaje automática relacionada con la forma en que los agentes deben tomar decisiones en un entorno con el fin de maximizar las recompensas que recibe. El aprendizaje de refuerzo es uno de los tres paradigmas básicos de aprendizaje automático, junto con el aprendizaje supervisado y el aprendizaje no supervisado. [5]

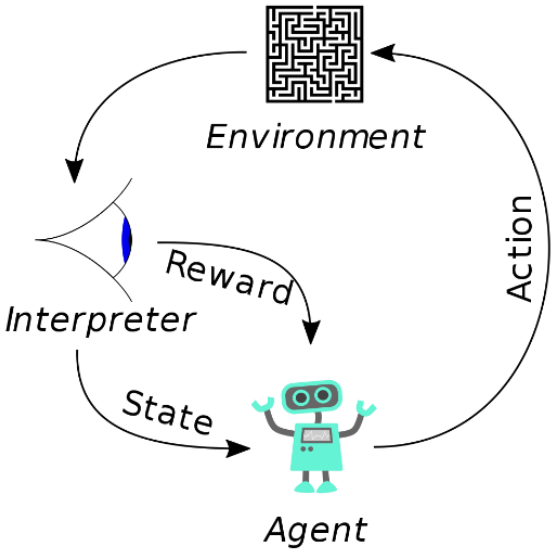

Figura 1.- Proceso de aprendizaje por refuerzo (fuente: By Megajuice - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=57895741g)

2.2. Redes Neuronales

Una red neuronal consta de unidades (neuronas), dispuestas en capas, que convierten un vector de entrada en alguna salida. Cada unidad toma una entrada, le aplica una función (a menudo no lineal) y, a continuación, pasa la salida a la siguiente capa. [6]

Generalmente, las redes se definen como feed-forward: una unidad alimenta su salida a todas las unidades de la capa siguiente, pero no hay retroalimentación a la capa anterior. Las ponderaciones se aplican a las señales que pasan de una unidad a otra, y son estas ponderaciones las que se ajustan en la fase de entrenamiento para adaptar una red neuronal al problema en cuestión.

El nombre viene de la idea de imitar el funcionamiento de las redes neuronales de los organismos vivos: un conjunto de neuronas conectadas entre sí y que trabajan en conjunto, sin que haya una tarea concreta para cada una. Con la experiencia, las neuronas van creando y reforzando ciertas conexiones para "aprender" algo que se queda fijo en el tejido.

2.2.1. Perceptrón

El perceptrón es la red neuronal más básica que existe de aprendizaje supervisado.

El funcionamiento del perceptrón es muy sencillo, simplemente lee los valores de entrada, suma todas las entradas de acuerdo a unos pesos y el resultado lo introduce en una función de activación que genera el resultado final. [7]

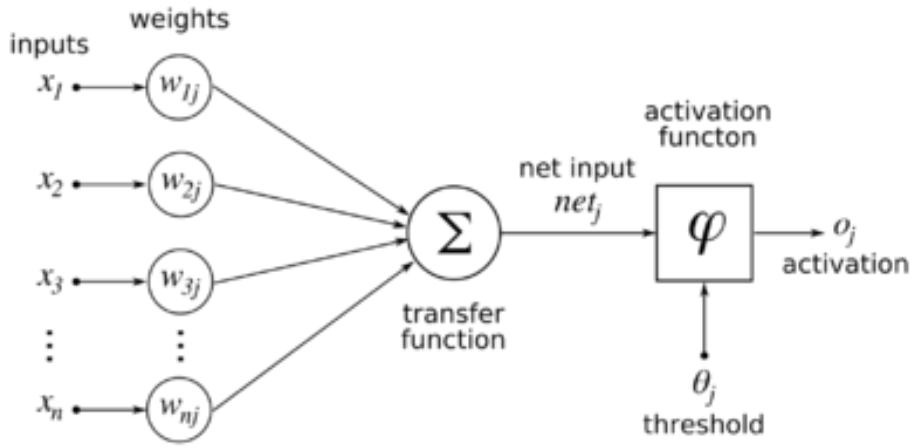

Figura 2.- Representación de un perceptrón (By Chrislb - created by Chrislb, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=224555)

El entrenamiento del perceptrón no es más que determinar los pesos sinápticos y el umbral que hagan que la entrada se ajuste lo mejor posible a la salida. Para la determinación de estas variables, se sigue un proceso adaptativo. Comenzando con valores aleatorios y progresivamente se van modificando según la diferencia entre los valores deseados y los calculados por la red, conocido como función de coste.

2.2.2. Perceptrón multicapa

El perceptrón multicapa consiste en el mismo modelo de perceptrón simple, pero con más neuronas formando una red de neuronas.

Una fila de neuronas se denomina capa y una red puede tener varias capas. La arquitectura de las neuronas en la red se llama a menudo topología de red.

El perceptrón multicapa es una red neuronal artificial (RNA) formada por múltiples capas, de tal manera que tiene capacidad para resolver problemas que no son linealmente separables, lo cual es la principal limitación del perceptrón (también llamado perceptrón simple). [8]

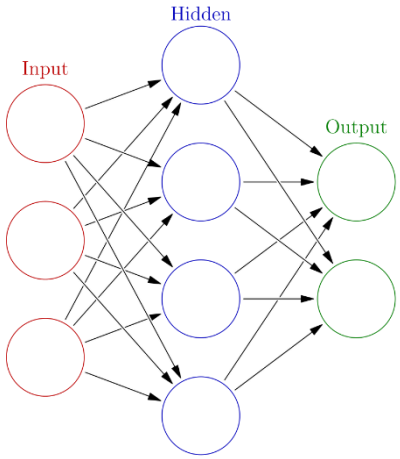

Figura 3.- Modelo de red neuronal simple (By Glosser.ca - Own work, Derivative of File:Artificial neural network.svg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=24913461)

Capas de entrada o visibles

La capa primera que toma la entrada del dataset se denomina capa visible, porque es la parte expuesta de la red. A menudo, una red neuronal se dibuja con una capa visible con una neurona por valor de entrada o columna en el conjunto de datos. Estas no son

neuronas como se describió anteriormente, sino que simplemente pasan el valor de entrada a través de la siguiente capa.

Capas ocultas

Las capas posteriores a la capa de entrada se denominan capas ocultas porque no se exponen directamente a la entrada. La estructura de red más simple es tener una sola neurona en la capa oculta que genera directamente el valor. [9]

Dados los aumentos en la potencia de cómputo y las bibliotecas eficientes, se pueden construir redes neuronales muy complejas, con multitud de nodos por cada capa oculta y multitud de capas ocultas. El aprendizaje profundo puede referirse a tener muchas capas ocultas en la red neuronal. Son profundos porque habrían sido inimaginablemente lentos para entrenar históricamente, pero pueden tomar segundos o minutos para entrenar utilizando técnicas y hardware modernos.

Capa de salida

La capa oculta final se denomina capa de salida y es responsable de generar un valor o vector de valores que correspondan al formato requerido para el problema. [9]

La elección de la función de activación en la capa de salida está fuertemente limitada por el tipo de problema que está modelando. Por ejemplo:

- Un problema de regresión puede tener una sola neurona de salida y la neurona puede no tener ninguna función de activación.
- Un problema de clasificación binaria puede tener una sola neurona de salida y utilizar una función de activación sigmoide para generar un valor entre 0 y 1 para representar la probabilidad de predecir un valor para la clase 1.
- Un problema de clasificación multiclase puede tener varias neuronas en la capa de salida, una para cada clase. En este caso se puede utilizar una función de activación softmax para generar una probabilidad de que la red prediga cada uno de los valores de clase.

2.2.3. Descenso de gradiente

El descenso de gradiente es un algoritmo de optimización usado para minimizar la función de coste de forma iterativa moviéndose en la dirección de descenso más pronunciado.

En el aprendizaje automático, utilizamos el descenso de gradiente para actualizar los parámetros de nuestro modelo. [10]

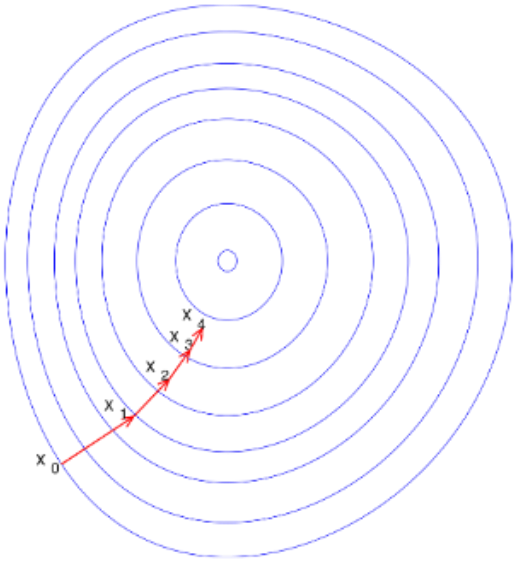

Figura 4.- Visualización 3D del algoritmo de descenso de gradiente (This file was derived from: Gradient descent.png:, Public Domain, https://commons.wikimedia.org/w/index.php?curid=20569355)

SGD por lotes

Los pesos en la red se pueden actualizar a partir de los errores calculados para cada ejemplo de entrenamiento y esto se denomina aprendizaje en línea. Puede resultar en cambios rápidos, pero también caóticos en la red.

Los errores se pueden guardar en todos los ejemplos de entrenamiento y la red se puede actualizar al final. Esto se llama aprendizaje por lotes y a menudo es más estable.

Normalmente, dado que los datasets son tan grandes y debido a eficiencias computacionales, el tamaño del lote se reduce a un número pequeño, como decenas o cientos de ejemplos.

2.2.4. Back Propagation

La propagación hacia atrás de errores o retro propagación (del inglés backpropagation) es un método de cálculo del gradiente utilizado en algoritmos de aprendizaje supervisado utilizados para entrenar redes neuronales artificiales. El método emplea un ciclo propagación – adaptación de dos fases. Una vez

que se ha aplicado un patrón a la entrada de la red como estímulo, este se propaga desde la primera capa a través de las capas siguientes de la red, hasta generar una salida. La señal de salida se compara con la salida deseada y se calcula una señal de error para cada una de las salidas. [11]

Las salidas de error se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. Sin embargo, las neuronas de la capa oculta solo reciben una fracción de la señal total del error, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total.

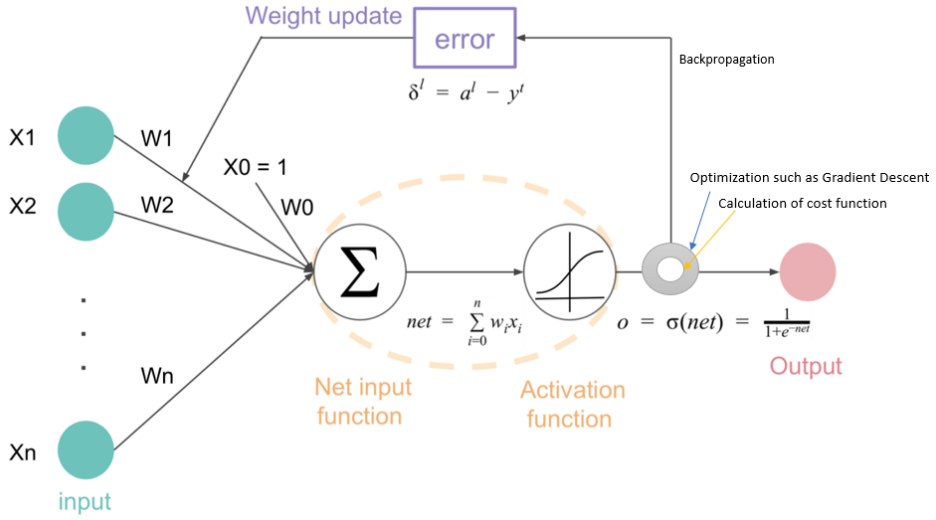

*Figura 5.- Método de backpropagation (fuente: https://miro.medium.com/max/1400/0*j8Tnd_tnc7vqdNtA.png)*

La importancia de este proceso consiste en que, a medida que se entrena la red, las neuronas de las capas intermedias se organizan a sí mismas de tal modo que las distintas neuronas aprenden a reconocer distintas características del espacio total de entrada. Después del entrenamiento, cuando se les presente un patrón arbitrario de entrada que contenga ruido o que esté incompleto, las neuronas de la capa oculta de la red responderán con una salida activa si la nueva entrada contiene un patrón que se asemeje a aquella característica que las neuronas individuales hayan aprendido a reconocer durante su entrenamiento.

2.3. Redes neuronales convolucionales (CNN)

Una red neuronal convolucional es un tipo de red neuronal artificial donde las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria (V1) de un cerebro biológico. [12]

Este tipo de red es una variación de un perceptrón multicapa, sin embargo, debido a que su aplicación es realizada en matrices bidimensionales, son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes, entre otras aplicaciones.

2.3.1. Arquitectura

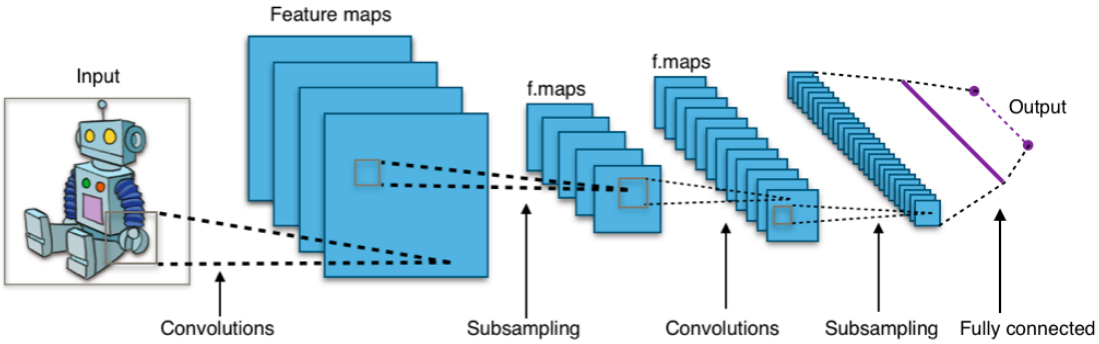

Figura 6.- Representación general de las capas internas de una red convolucional (By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45679374)

Capa de entrada

Son las entradas directas de la imagen, los valores de cada píxel conforman el número total de neuronas de la capa inicial de entrada. No se lleva a cabo ninguna operación per se.

Capa de convolución

En la convolución se realizan operaciones de productos y sumas entre la capa de partida y los n filtros (o kernel) que genera un mapa de características. Las características extraídas corresponden a cada posible ubicación del filtro en la imagen original.

La ventaja es que el mismo filtro sirve para extraer la misma característica en cualquier parte de la entrada, con esto que consigue reducir el número de conexiones y el

número de parámetros a entrenar en comparación con una red multicapa de conexión total.

Convolución

El proceso de convolución son una serie de operaciones matemáticas que se realizan entre la imagen a tratar y un filtro o kernel.

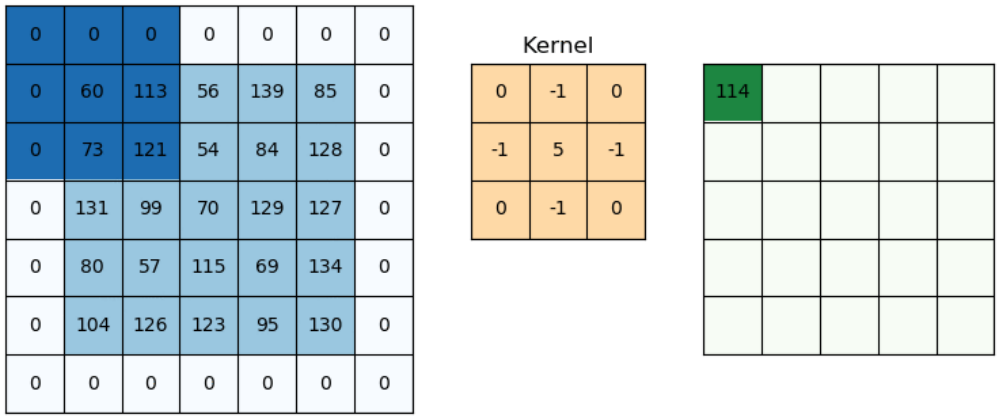

Figura 7.- Convolución de una imagen con un kernel (3,3) (fuente: https://i.ibb.co/Vvp6FRz/nagesh-cnn-intro-5.gif)

La convolución de la imagen con el kernel reduce el tamaño de la imagen original. Es un aspecto positivo si lo que queremos es reducir el tamaño de la entrada para aligerar carga computacional, sin embargo, en capas de abstracción internas, la imagen no debería disminuir en tamaño, ya que corremos el peligro de que todas las características únicas que definen la imagen se pierdan. Para solucionar este tipo de problema, se hace uso del zero padding.

Zero padding

El padding no es más que rodear la imagen con ceros, para que conserve el aspecto original. De este modo, una vez reducida la imagen y detectadas características fundamentales de la misma, no se reduce el tamaño y se puede definir mejor una característica que se haya encontrado.

Capa de extracción

En la capa de extracción se reduce la cantidad de parámetros al quedarse con los valores máximos o promedios de un kernel, comúnmente de 2x2, sobre la imagen.

La forma de reducir parámetros se realiza mediante la extracción de estadísticas como el promedio o el máximo de una región fija de la imagen, perdiendo precisión, pero mejorando su compatibilidad. Cuando se reducen los parámetros mediante la función máxima, se le denomina MaxPooling y cuando se realiza por valores medios, Average Pooling.

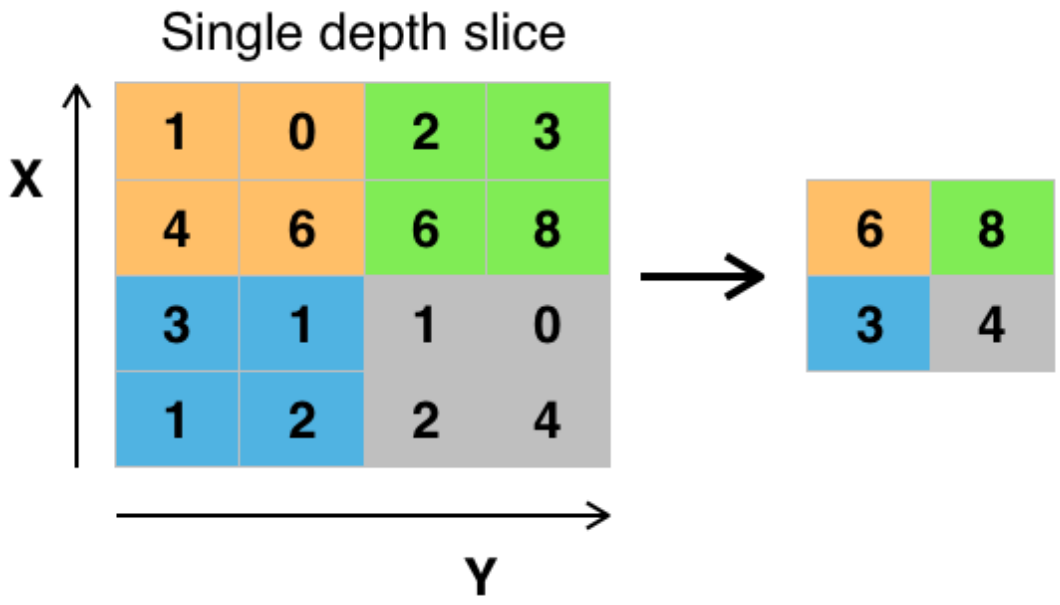

Figura 8.- Operación MaxPooling (By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45673581)

Max Pooling también funciona como un supresor de ruido. Descarta activaciones ruidosas y realiza de-noising al reducir la dimensionalidad de la imagen. Por otro lado, Average Pooling simplemente realiza reducción dimensional como forma de reducir el ruido global, concluyendo con que MaxPooling genera resultados mejores. [13]

Capas completamente conectadas

El final de las capas convolucional y de reducción, se suele utilizar capas completamente conectadas en la que cada pixel se considera como una neurona separada al igual que en un perceptrón multicapa tras aplanar el array bidimensional a uno de una dimensión conteniendo el valor de cada pixel de la imagen.

Agregar una capa totalmente conectada es una forma (normalmente) barata de aprender combinaciones no lineales de las características extraídas de la salida de la capa convolucional. [13]

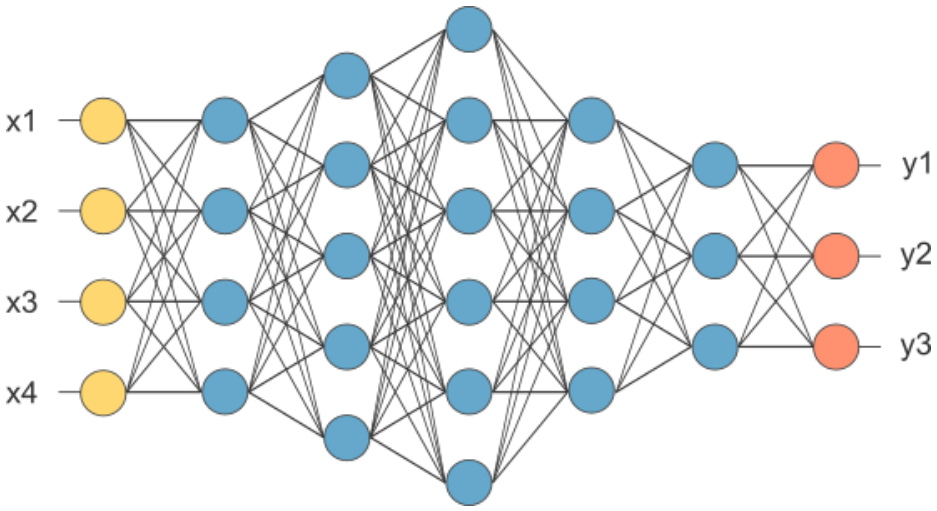

*Figura 9.- Capa densa (fuente: https://miro.medium.com/max/1108/1*Mw6LKUG8AWQhG73H1caT8w.png)*

La última capa de esta red es una capa clasificadora que tendrá tantas neuronas como el número de clases a predecir, empleando una función de activación como softmax o sigmoide.

2.4. Funciones de activación, optimizadores y regularización

2.4.1. Función no lineal ReLU

Durante mucho tiempo, la activación predeterminada a utilizar fue la función de activación sigmoide. Más tarde, fue la función de activación tanh. Para las redes

neuronales modernas de aprendizaje profundo, la función de activación predeterminada es la función de activación lineal rectificada. [14]

La mayoría de los documentos que logran resultados de última generación describirán una red utilizando ReLU. Por ejemplo, en el artículo de 2012 de Alex Krizhevsky, et al. *ImageNet Classification with Deep Convolutional Neural Networks,"* [15] los autores desarrollaron una profunda red neuronal convolucional con activaciones ReLU que lograron resultados de última generación en el conjunto de datos de clasificación fotográfica de ImageNet. [16]

RELU significa Unidad Lineal Rectificada para una operación no lineal. La salida es:

$$f(x) = \max(0, x) \tag{1}$$

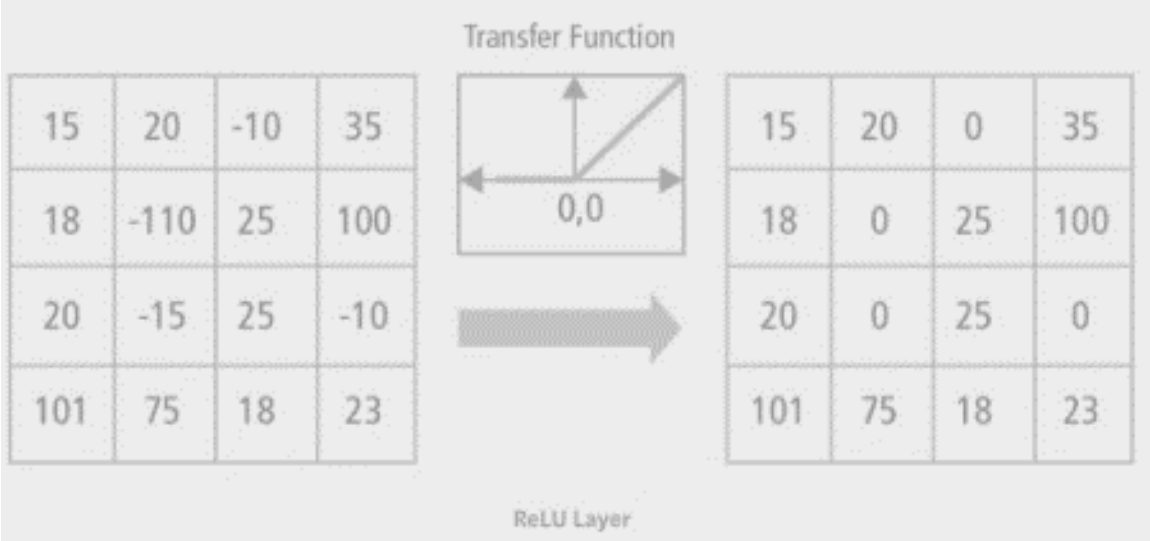

*Figura 10.- Operación de la función de activación ReLU (fuente: https://miro.medium.com/max/898/1*gcvuKm3nUePXwUOLXfLIMQ.png)*

El propósito de ReLU es introducir la no linealidad en nuestra red neuronal, y convertir aquellos valores negativos en cero.

2.4.2. Softmax

En matemáticas, la función softmax, o función exponencial normalizada,1:198 es una generalización de la Función logística. Se emplea para "comprimir" un vector K-

dimensional, de valores reales arbitrarios en un vector K-dimensional de valores reales en el rango [0, 1]. [17]

La función está dada por:

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_i}} \quad (2)$$

La función softmax es empleada en varios métodos de clasificación multiclase tales como Regresión Logística Multinomial, análisis discriminante lineal multiclase, clasificadores Bayesianos ingenuos (naive Bayes), y Redes Neuronales Artificiales.

2.4.3. Optimizadores

α = Ratio de aprendizaje

W = **Matriz de pesos**

ΔW = **Derivada de la matriz de pesos**

Stochastic Gradient Descend (SGD)

Es el método básico de optimización de redes neuronales y otros algoritmos de machine learning. La actualización de pesos está definida como sigue:

$$W = W - \alpha * \Delta W \quad (3)$$

En este caso, la ratio de aprendizaje **α** es fijo, es decir, no cambia nunca durante el entrenamiento. [18]

Adagrad

Este método adaptativo ajusta **α** a los parámetros de la red, donde actualizaciones más grandes se llevan a cabo sobre parámetros que varían infrecuentemente, mientras que los que varían más seguido, reciben un ajuste más pequeño. [18]

$$C = C + \Delta W^2 \tag{4}$$

$$W = W - \frac{\alpha * \Delta W}{\sqrt{C + \epsilon}} \tag{5}$$

C es un acumulador que mantiene la suma de los cuadrados de los gradientes (derivadas), por cada parámetro. C nos permite actualizar cada parámetro con respecto a la magnitud de su cambio.

Posteriormente, dividimos $\boldsymbol{\alpha}$***ΔW** entre la raíz cuadrada de C, añadiendo un pequeño valor, ε, para evitar divisiones entre 0.

Adadelta

Es una mejora a Adagrad, la cual busca reducir el decrecimiento monotónico del learning rate del que sufre éste debido a la naturaleza del acumulador, C.

La formulación es la misma, Adagrad calcula el acumulador para cada lote, tomando en cuenta todo el historial de gradientes cuadrados, Adadelta lo hace sólo para un pequeño número de gradientes cuadrados anteriores. Así, lo que este optimizador hace es calcular el promedio del historial de gradientes. [18]

RMSprop

Al igual que Adadelta, busca corregir los efectos negativos de la acumulación, al convertirlo en un promedio movible exponencial con pesos.

$$C = C + d * C * (1 - d) * \Delta W^2 \tag{6}$$

$$W = W - \frac{\alpha * \Delta W}{\sqrt{C + \epsilon}} \tag{7}$$

El nuevo parámetro, **d**, conocido como **decaying rate**, o tasa de descomposición/decadencia, es un hiperparámetro típicamente fijado en 0.9, el cual hace que las entradas anteriores en el cache influyan considerablemente menos que las nuevas. [18]

Adam

Adaptive Moment Estimation (Estimación Adaptativa de Momentos), el cual es, en pocas palabras, RMSProp con momentum.

$$m = \beta_1 * m + (1 - \beta_1) * \Delta W \quad (8)$$

$$v = \beta_2 * v + (1 - \beta_2) * \Delta W^2 \quad (9)$$

$$W = W - \frac{\alpha * m}{\sqrt{v + \epsilon}} \quad (10)$$

m y **v** representan los dos momentos, siendo **m** el que modela la media de los gradientes a lo largo del tiempo, mientras que **v** hace lo mismo con la varianza. **β1** es igual, en la mayoría de los casos, a 0.9, mientras que **β2** casi siempre se fija en 0.99.

Adam puede ser visto como una combinación de RMSprop y SGD con momento. Utiliza los gradientes al cuadrado para escalar la tasa de aprendizaje como RMSprop y aprovecha el momento mediante el uso de la media móvil del gradiente en lugar de gradiente en sí como SGD con momento. [18]

2.4.4. Regularizadores

La regularización no es más que añadir un término de penalización para controlar la complejidad del modelo —evitar el exceso de ajuste— utilizando ese término de penalización. Por lo tanto, el modelo generaliza mejor y aumentando así el rendimiento del modelo.

Dropout

La deserción es una técnica de regularización para los modelos de redes neuronales propuestos por Srivastava, et al. en su artículo de 2014 *Dropout: A Simple Way to prevent Neural Networks from Overfitting.* [19]

Es una técnica de regularización que consiste en descartar ciertas neuronas de manera aleatoria durante la fase de entrenamiento, cancelando temporalmente la información que antes aportaba al conjunto de la red.

A medida que una red neuronal aprende, los pesos de las neuronas se asientan en su contexto dentro de la red. Dichas neuronas proveen a la red de su especialización a las

neuronas vecinas que se convierten dependientes de esta especialización, y si se lleva demasiado lejos puede resultar en un modelo frágil demasiado especializado para los datos de entrenamiento. Esta dependencia del contexto de una neurona durante el entrenamiento se refiere a co-adaptaciones complejas.

Podemos imaginar que, si las neuronas se dejan al azar fuera de la red durante el entrenamiento, otras neuronas tendrán que intervenir y hacer predicciones para las neuronas que faltan. Esto se cree que da lugar a múltiples representaciones internas independientes que son aprendidas por la red.

El efecto es que la red se vuelve menos sensible a los pesos específicos de las neuronas. Esto a su vez da como resultado una red que es capaz de una mejor generalización y es menos probable que sobreajuste los datos de entrenamiento.

2.5. Trabajos realizados

An Algorithm for Japanese Character Recognition [20]

Este estudio propone el estudio de caracteres del alfabeto japonés **hiragana** por un algoritmo de topología geométrica.

Este algoritmo se basa en la identificación del centro de gravedad y su tamaño, traslación y rotación invariable. Además, en el centro de gravedad, se han utilizado puntos de referencia basados en topología, como puntos de conjunción que enmascaran la intersección de bucles cerrados y trazos múltiples, así como puntos finales para calcular los centros de gravedad de estos puntos ubicados en los cuadrantes individuales de los círculos que encierran los caracteres. Después de los pasos iniciales de preprocesamiento como notarización, cambio de tamaño, recorte, eliminación de ruido, sincronización, el número total de puntos de conjunción, así como el número total de puntos finales se calculan y almacenan. El personaje se rodea y se divide en cuatro cuadrantes. Se calculan el centro de gravedad (engranaje) de todo el carácter, así como los engranajes de cada uno de los cuatro cuadrantes, y se calculan y almacenan las distancias euclidianas de los puntos de conjunción y fin en cada uno de los cuadrantes con los engranajes. Los valores de estas cantidades se calculan para las imágenes de destino y de plantilla y se computa

una coincidencia con el carácter que tiene la distancia euclidiana mínima. La precisión media obtenida es del 94,1 %.

A neural framework for online recognition of handwritten Kanji characters [21]

El objetivo de este estudio es proponer un marco eficiente y rápido para el reconocimiento de caracteres Kanji, trabajando en tiempo real durante su escritura. Investigaciones anteriores sobre reconocimiento en línea de caracteres escritos a mano utilizaban un gran conjunto de datos que contenía muestras de caracteres escritos por muchos escritores. Nuestro estudio presenta una solución que logra resultados finos, utilizando un pequeño conjunto de datos que contiene una sola muestra para cada carácter Kanji de un solo escritor. El sistema propuesto analiza y clasifica los tipos de trazo que aparecen en un Kanji y luego lo reconoce. Para ello, utilizamos una red neuronal convolucional y un diccionario jerárquico que contiene definiciones Kanji. Además, comparamos los histogramas de Kanjis para resolver el problema de distinguir el carácter que tiene el mismo número de trazos del mismo tipo, pero dispuestos en una posición diferente en relación entre sí. El marco propuesto fue validado experimentalmente en Kanjis escritos a mano en línea por principiantes y estudiantes avanzados. La precisión lograda de hasta un 89% indica que puede ser una solución valiosa para aprender Kanji por principiantes.

Aplicación de Deep Learning a la Enseñanza de la Escritura Japonesa [22]

La enseñanza de la escritura de idiomas es un paso fundamental en el proceso de aprendizaje de un idioma. Pese a esto, las aplicaciones más conocidas dedicadas a los idiomas no se focalizan en este ámbito. Hay muchas aplicaciones pequeñas que se especializan únicamente en la escritura de un idioma mediante el uso de plantillas caligráficas correctoras del trazado. En este trabajo de fin de grado se propone un nuevo método de enseñanza de la escritura: el uso de redes de neuronas convolucionales aplicadas al reconocimiento de imágenes de caracteres. Los caracteres estudiados son los respectivos de los silabarios japoneses hiragana y katakana. Tras una experimentación previa con otros dominios como CIFAR10, Dogs vs. Cats y FERC, con el conjunto de datos de caracteres ETL Character Database se crean dos modelos de predicción, uno por cada silabario, obteniendo un 99,23 % de acierto en el conjunto de validación en el modelo de katakana y un 99,61 % en el de hiragana.

Aplicaciones

Existen multitud de aplicaciones de reconocimiento de kanji orientadas a hacer ejercicios tipo test para determinar cuántos kanjis sabes, y para que aprendas nuevos. Es una estrategia interesante si quieres poner a prueba tus conocimientos. Las aplicaciones determinan que kanji escribe el usuario por método del número de trazos dibujados, y el orden de los mismos. Algunos ejemplos son:

Kanji Recognizer [23], *Estudio Kanji japonés - 漢字 学習* [24], *KanjiTree Japonés* [25]

Para el reconocimiento, debes dibujar correctamente el orden de los trazos y de una forma específica en el canvas que se ofrece.

Una diferencia clave en el desarrollo de este trabajo es eliminar por completo la necesidad de saber el orden de los trazos cuando se comienza a aprender, ya que, es poco probable que se sepa el orden correcto y el número de trazos de un símbolo. Si bien es cierto que todas las aplicaciones, sumando la de este proyecto, contienen información individual del kanji donde ya sí se explica que orden se debe seguir, para que el aprendizaje sea correcto según la tradición de escritura japonesa.

3. Implementación

3.1. Técnicas de clasificación MLP y CNN

Existen multitud de librerías dedicadas a desarrollar clasificadores de imágenes basados en modelos de perceptrón multicapa y redes neuronales convolucionales.

Algunas de las herramientas a nuestra disposición hacen uso del modelo del perceptrón multicapa. Son de las primeras técnicas empleadas en problemas de regresión lineal y no lineal y son un buen punto de salida para crear un modelo sencillo, ya que, los modelos basados en MLP pueden aproximar un problema no lineal con bastante precisión como es el caso de clasificación de imágenes por grupos.

Es un modelo que, aun siendo sobrepasado en uso por las redes neuronales convolucionales, da resultados no muy diferentes en comparación, como se demuestra en la clasificación de imágenes de MNIST.

Durante el desarrollo de este trabajo se comprobará dicho rendimiento en comparación con las redes neuronales convolucionales, actualmente el algoritmo más usado para visión de ordenador y ampliando el área del procesamiento del lenguaje. Es un sistema muy robusto, aunque más costoso computacionalmente que los modelos MLP.

3.2. Herramientas de desarrollo

Para el primer modelo de red neuronal, se hará uso de la API de sklearn en Python el módulo de redes neuronales que incluye un modelo de perceptrón multicapa para problemas de regresión y clasificación: MLPClassifier.

Existen librerías que permiten el uso de funciones para crear modelos de redes convolucionales con relativa sencillez y de entre ellas destacan Theano y Tensorflow.

Para el desarrollo de la red neuronal convolucional, se empleará Tensorflow, sin embargo, se utilizará una API para usar funcionalidades de Tensorflow de manera más sencilla con Keras. Y dado que el cálculo computacional que viene de serie con el entrenamiento de una red convolucional, se hará empleo de librerías de Nvidia cuDNN incluidas en la API de Nvidia CUDA Toolkit.

3.2.1. Ionic

El framework Ionic es un kit de herramientas para UI (user interface) de código libre enfocado a móviles para construir aplicaciones web multiplataforma. [26]

Ofrece librerías especialmente optimizadas para dispositivos móviles, reconocimiento de gestos, componentes reusables y código que compila en todas las plataformas.

También ofrece la posibilidad de integrar tres de los framework más populares hoy día como: Angular, React y Vue.

Incluso da la libertad de escribir tu código en vanila JS o JavaScript, sin necesidad del uso de ningún framework.

3.2.2. Angular

Angular es el framework elegido para desarrollar la aplicación de teléfono ya que es uno de los más antiguos y completos que existen.

Angular (comúnmente llamado Angular 2+ o Angular 2) es un framework para aplicaciones web desarrollado en TypeScript, de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles. [27]

3.2.3. CUDA Toolkit

NVIDIA CUDA Toolkit proporciona un entorno para crear aplicaciones aceleradas por GPU de alto rendimiento. El kit de herramientas incluye bibliotecas aceleradas por GPU, herramientas de depuración y optimización, un compilador de C/C++ y una biblioteca en tiempo de ejecución para implementar la aplicación. [28]

Las bibliotecas CUDA aceleradas por GPU permiten la aceleración de entrada en varios dominios, como el álgebra lineal, el procesamiento de imágenes y vídeos, el aprendizaje profundo y el análisis de gráficos.

3.2.4. Theano

Theano es una biblioteca de Python que te permite definir, optimizar y evaluar expresiones matemáticas que implican matrices multidimensionales de manera eficiente. [29]

3.2.5. Tensorflow

Tensorflow es una plataforma de código abierto de extremo a extremo para el aprendizaje automático. Cuenta con un ecosistema integral y flexible de herramientas, bibliotecas y recursos de la comunidad que les permite a los investigadores impulsar un aprendizaje automático innovador y, a los desarrolladores, compilar e implementar con facilidad aplicaciones con tecnología de ML (machine learning). [30]

- Compilación sencilla de modelos
- Producción de ML sólido en cualquier parte
- Importante experimentación para la investigación

3.2.6. Keras

Keras es una API diseñada para seres humanos, no para máquinas. Keras sigue las prácticas recomendadas para reducir la carga cognitiva: ofrece API coherentes y sencillas, minimiza el número de acciones de usuario necesarias para casos de uso comunes y proporciona mensajes de error claros y procesables. También cuenta con una amplia documentación y guías para desarrolladores. [31]

3.2.7. Flask

Flask es un "micro" Framework escrito en Python y concebido para facilitar el desarrollo de Aplicaciones Web bajo el patrón MVC. [32]

El patrón MVC es una manera o una forma de trabajar que permite diferenciar y separar lo que es el modelo de datos (los datos que van a tener la App que normalmente están guardados en BD), la vista (página HTML) y el controlador (donde se gestiona las peticiones de la app web).

- Proporciona una estructura del proyecto, es decir, todas las Apps que estén construidas con **Flask** van a tener los mismos elementos y los mismos ficheros.
- No se necesita una infraestructura con un servidor web para probar las aplicaciones sino de una manera sencilla se puede correr un servidor web para ir viendo los resultados que se van obteniendo.
- Es fácil encontrar librerías adaptadas al Framework.

- Flask es Open Source

3.2.8. Entorno de programación

Para la vista de usuario se empleará un framework de industria llamado Ionic, con el que se dará formato visual a la interfaz de usuario de la aplicación de teléfono. Las ventajas de usar un framework es que se agiliza el desarrollo de los componentes esenciales de la aplicación de forma notable, para obtener un boceto del estilo visual y funcional que tendrá la aplicación, además de ser multiplataforma.

En cuanto a la parte de servidor, será desarrollado en el lenguaje de programación Python ya que es un lenguaje orientado a desarrollo de datos y cuenta con librerías con funcionalidades muy útiles a la hora de tratar con matrices de datos. En esta ocasión se usará Flask para facilitar el enrutamiento con la aplicación de teléfono.

Tanto Ionic como el servidor de Python se desarrollarán en un entorno de desarrollo integrado (IDE) denominado VSCode de Microsoft.

3.3. Obtención de datos

Sin duda el paso más crítico a la hora de crear cualquier IA es con qué conjunto de datos vas a entrenar la red neuronal, ya sea esta un clasificador multicapa o una red convolucional.

Contar con un conjunto de datos bien definido y variado es clave para que la red neuronal esté preparada para predecir eventos futuros totalmente nuevos para ella, ya que si es entrenada con datos muy similares entre sí para cada clase, o si bien en pequeño volumen de datos, se corre el riesgo de que la red neuronal acabe prediciendo a la perfección los datos con los que fue entrenada y sin embargo a la hora de introducir nueva información para su predicción, esta sea incorrecta o incluso aleatoria.

Debido a la dificultad de encontrar un dataset que satisfaga las necesidades de este trabajo, se ha optado por una opción más costosa en cuanto a tiempo invertido, ya que el dataset se creará desde cero.

En aras de optimizar el tiempo dedicado a la creación del dataset, se almacenarán 500 muestras por cada kanji por imposibilidad de equipamiento hardware ya que los recursos de memoria RAM son muy limitantes.

Se ha decidido que se dibujarán los primeros 80 kanjis que se estudian en los primeros años de primaria en Japón recogidos en la lista Kyōiku kanji. [33]

Dado que la aplicación tiene unas características fijas, el canvas donde se dibujan los kanjis no varía de dispositivo en dispositivo, siendo siempre de 300x300 píxeles. Por tanto, los datos que se almacenarán para el entrenamiento de la red neuronal serán directamente obtenidos de la aplicación.

3.3.1. Diseño de la aplicación

La aplicación está pensada para ser de máxima utilidad, sin pantalla principal, sin distracciones en general. Se ha diseñado para que el usuario entre para resolver su duda de la forma más eficaz y rápida posible.

A continuación, se muestra un diagrama del flujo de trabajo dentro de la aplicación:

Figura 11.- Esquema de flujo de trabajo en la aplicación

Se definen dos pantallas para interactuar con la aplicación. Una de ellas es la zona de dibujo para el reconocimiento del kanji, mientras que la segunda es un diccionario de kanji, donde puedes consultar kanjis por niveles o grados.

Un aspecto fundamental en el que está orientado cómo el usuario dibuja, es que no está restringido a conocer la forma de los trazos ni el total de trazos que tenga el kanji que desea reconocer. El total de las aplicaciones móviles vistas en el apartado 2.5., tienen el condicionante de ser eficaces siempre y cuando se dibuje correctamente el kanji, siguiendo su forma tradicional en el trazo. Es una buena práctica, pero cuando orientas una aplicación a principiantes del idioma, no se espera que conozcan en detalle el orden correcto de dibujar cada kanji.

Se ha optado por dar libertad al usuario en cuanto a dibujo se refiere para que sea la red neuronal la que haga toda la carga de trabajo y ser así capaz de reconocer el kanji que se ha dibujado.

Para dibujar contamos con un canvas de 300x300 píxeles como se muestra a continuación:

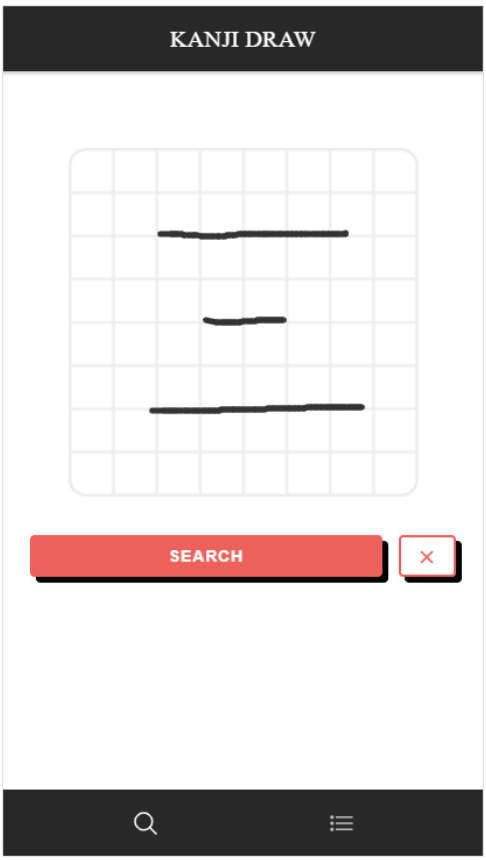

Figura 12.- Interfaz principal de la aplicación.

Se observa que la interfaz consta de un canvas fijo en cualquier dispositivo, para mantener la coherencia a la hora de prepocesar las imágenes y dos botones, uno por ahora para guardar la imagen en el dispositivo y que luego servirá para hacer uso del modelo entrenado para saber de qué kanji se trata, y otro simplemente para resetear el canvas por si el dibujo ha salido extremadamente mal.

La aplicación consta también de un diccionario para consultar kanjis por nivel de estudio en grados escolares.

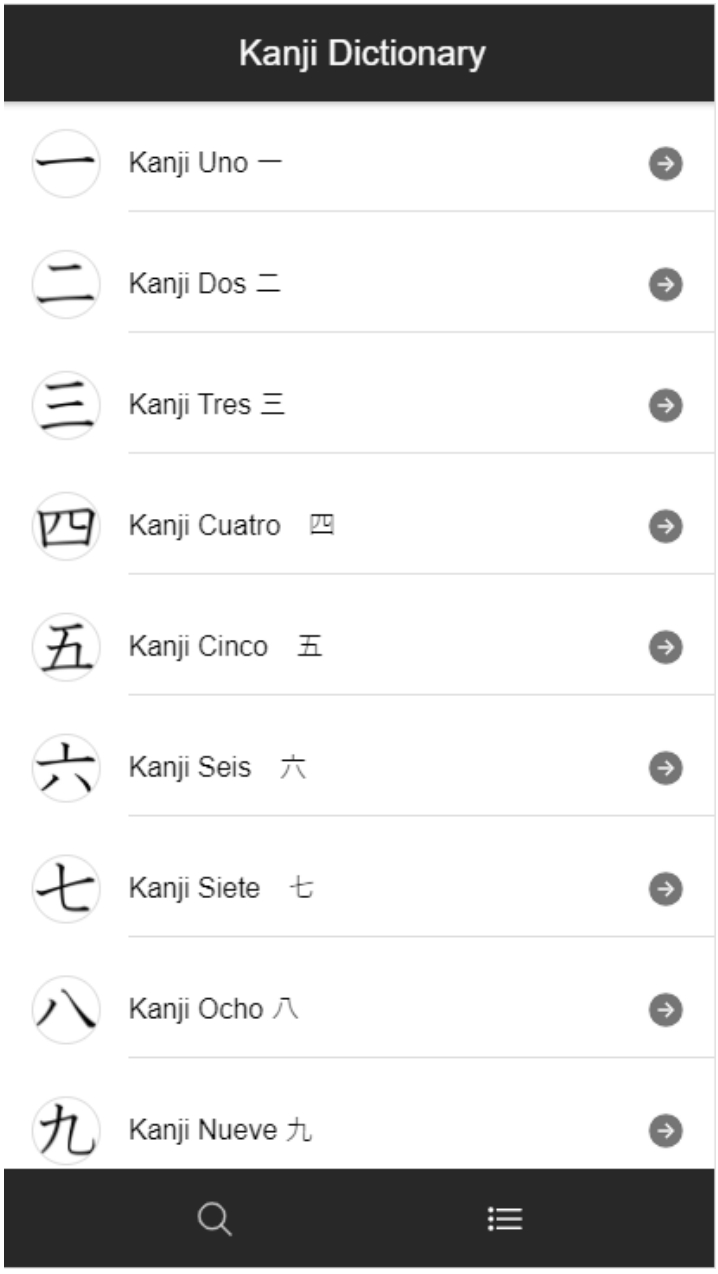

Figura 13.- Diccionario kanji

Cada kanji contiene información detallada de su uso y pronunciación en su ficha individual, donde se puede consultar también el orden del trazo, como está construido, y varios ejemplos de uso. Una vez se ha realizado el dibujo para reconocer, la aplicación puede mostrar dos posibles pantallas como se expone en la figura 11. Un listado de probabilidades es mostrado siempre que el modelo no esté seguro en más de un 98% de que el kanji dibujado corresponde a la clase predicha.

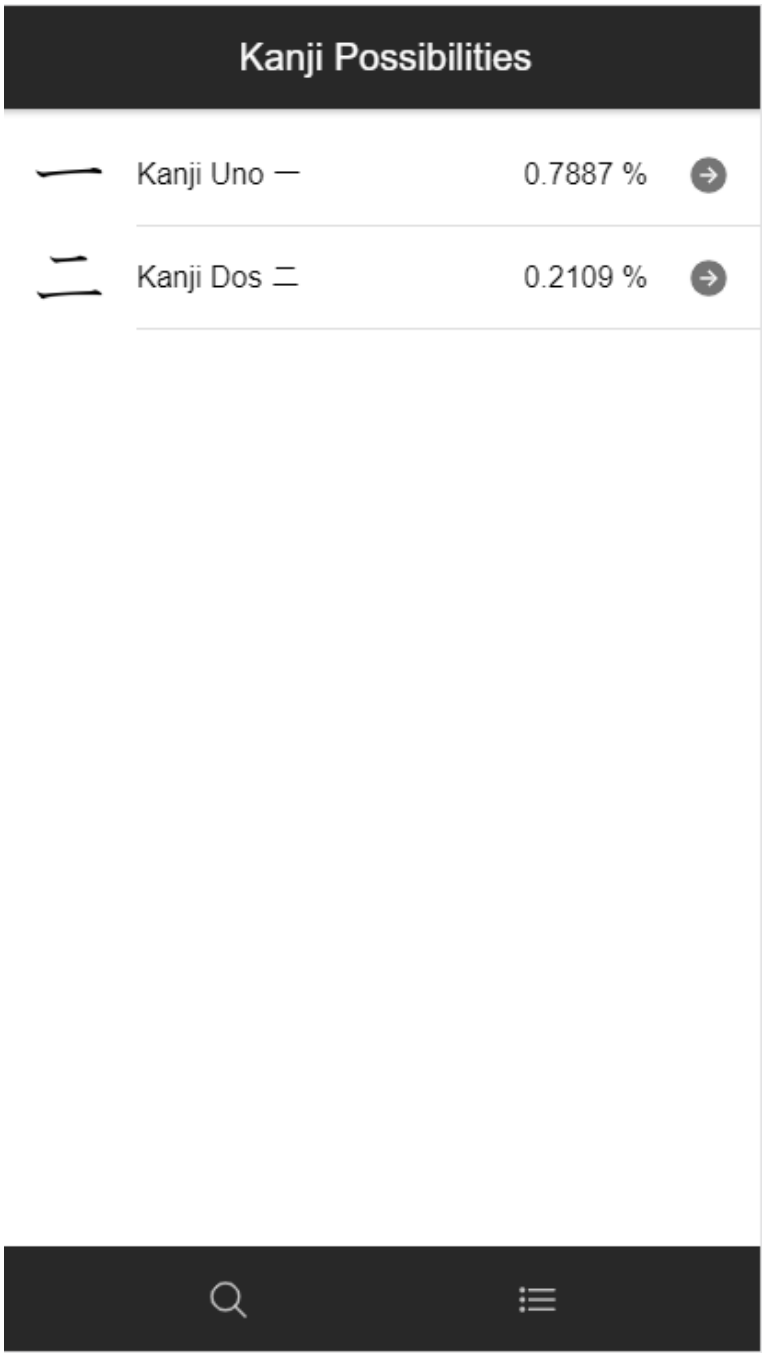

Figura 14.- Listado de posibilidades

Si la red neuronal cree en más de un 98% que el Kanji dibujado corresponde a la clase predicha, Se muestra directamente la ficha de información.

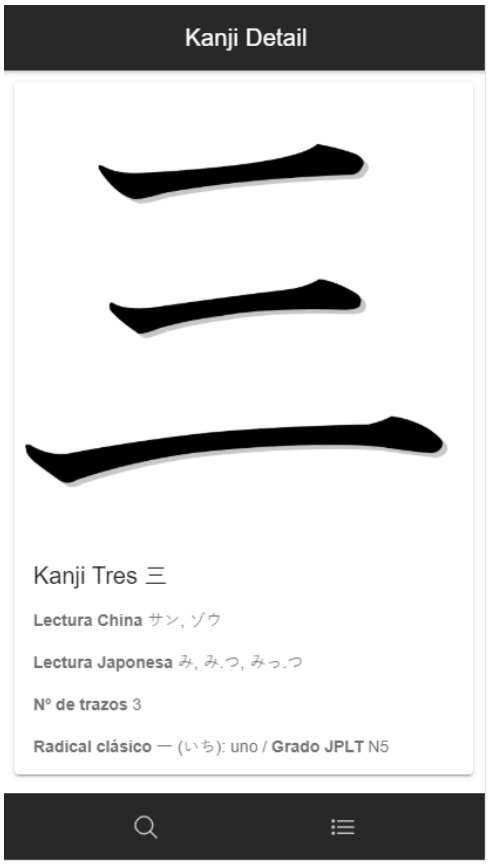

Figura 15.- Detalle del kanji reconocido

3.3.2. Guardado de las imágenes

El guardado de las imágenes se realiza en el mismo dispositivo empleando la interfaz de la figura 11. Si bien es poco eficiente, es más cómodo que crear toda una base de datos para las imágenes, aunque a más kanjis nuevos añadidos, la necesidad de incorporar una se hace patente.

Una vez dispongamos de suficientes kanjis para hacer pruebas, todas las carpetas de datos se volcarán a un ordenador para comenzar su tratamiento previo al entrenamiento.

3.3.3. Preprocesado de las imágenes

Una vez tenemos suficientes imágenes para comenzar a entrenar los modelos, primero se debe crear un formato que todas las imágenes van a cumplir. Para minimizar el tiempo de entrenamiento total, es necesario disminuir el ancho y alto de la imagen inicial y se ha

decidido reducirlo a 64x64 píxeles, delimitando el espacio requerido por el dibujo y descartando el espacio sobrante.

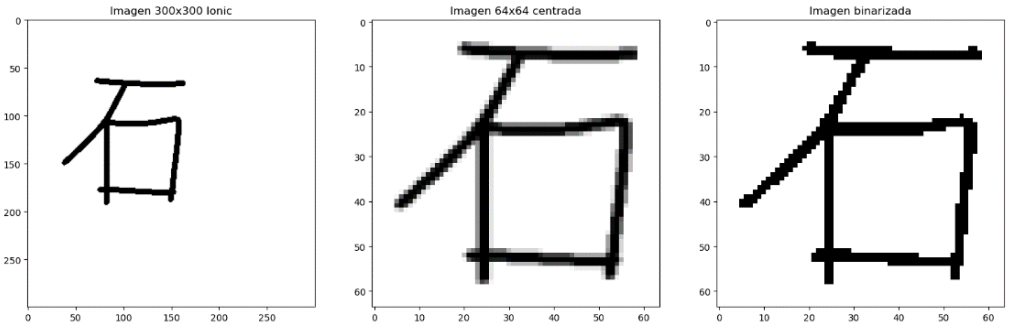

Figura 16.- Reescalado de la imagen original a un tamaño de 64x64 píxeles y binarizada.

Obsérvese como en la imagen de la izquierda el alto y ancho son de 300x300 píxeles, mientras que en la imagen del centro el total es de 64x64 píxeles, con un margen del 10% a cada lado de píxeles blancos. Esto consigue uniformidad a la hora de tratar con dibujos que se han desviado del centro o que se han dibujado pequeños en comparación con el total del canvas. Por último, se ha binarizado la imagen con el método OTSU. [34]

Todas las imágenes se han guardado en extensión PNG, sin embargo, el formato que recibirá la red neuronal será blanco y negro de un solo canal, en formato de coma flotante de 32 bits, normalizado entre [0,1].

Para la conversión de imágenes se ha empleado el paquete opencv-python en la última versión disponible. [35]

3.3.4. Creación del Dataset

Para completar el proceso previo a la fase de entrenamiento, es necesario agrupar todas las imágenes de forma matricial en un array numpy. [36]

Dado que las redes neuronales que vamos a entrenar entran dentro de la categoría de aprendizaje supervisado, es necesario un segundo array de valores para etiquetar cada una de las diferentes clases de kanjis que existen en el conjunto de imágenes preprocesadas, a fin de determinar la función de coste para actualizar la precisión con la que la red neuronal actuará sobre un caso real.

Los ficheros sobre los que se realizará el entrenamiento están compuestos de 500 muestras de cada kanji, lo que hace un total de 40.000 muestras, intercaladas en orden cada 80 ciclos y, en cuanto al array de etiquetas, se ha numerado del 0 al 79.

3.3.5. Desarrollo modelo Clasificador MLP

Como ya se ha comentado, el primer prototipo en el que realizar pruebas proviene de la API de Sklearn. Uno de sus varios módulos de redes neuronales es el Clasificador MLP (Multi-Layer Perceptron).

El clasificador MLP entrena iterativamente ya que en cada paso de tiempo se calculan las derivadas parciales de la función de pérdida con respecto a los parámetros del modelo para actualizar los parámetros.

Esta implementación funciona con datos representados como matrices numpy o matrices scipy dispersas de valores de punto flotante.

Algunos de los parámetros a tener en cuenta son, las capas ocultas del modelo, la función de activación para las capas ocultas y el optimizador de los pesos de la red.

3.3.6. Desarrollo del modelo CNN con Keras

El modelo para la red neuronal convolucional será implementado haciendo uso de la API de Keras para comunicarse con el back-end de Tensorflow. Ayudado por las librerías de Nvidia cuDNN para aprovechar al máximo el rendimiento de la tarjeta gráfica que se use que, en este caso, será una tarjeta RTX 2070 con 8GB GDDR6 de memoria.

3.4. Fase de entrenamiento

Una vez tenemos datos con los que empezar a entrenar las redes neuronales, es momento de crearlas. Como se ha explicado anteriormente, se van a crear dos modelos para comparar eficiencia entre ambos.

Ambos modelos dispondrán del mismo Dataset, conteniendo 80 kanjis con 500 muestras por kanji, separados en grupos para entrenar y testar de tamaños 400 y 100 respectivamente.

En total disponemos de 40.000 imágenes: 32.000 muestras para entrenar y 8.000 para testar la precisión de los modelos.

3.4.1. Clasificador MLP

Se han configurado distintas redes para poder comparar la precisión de cada una, evaluando en cada caso que es necesario para aumentar la ratio de aciertos e intentar que no haya un sobreajuste de la red.

Comenzando describiendo el modelo que ofrece sklearn, la librería en uso se llama MLPClassifier. Contiene una red neuronal con los parámetros por defecto ya definidos en un estado que se considera óptimo. Esto son:

- El número de capas ocultas son solo una, con 100 nodos
- La función de activación que emplea es la función ReLU
- El optimizador de pesos Adam
- Una ratio de entrenamiento constante a 0.001
- Uso de momento y nestervovs_momentum con parámetros beta_1 = 0.9 y beta_2 = 0.999

En la sección de análisis de resultados se profundiza en las gráficas para cada comparación que se define a continuación.

La primera de ellas comienza con el primero de los parámetros, el número de capas ocultas. Se han creado 6 modelos, uno de ellos el modelo estándar definido arriba, y tres más con diversas con diversos niveles de capas ocultas. El objetivo principal es determinar cuál de ellas produce un resultado significativo en cuanto a precisión se refiere.

Una vez definido el número de capas a usar, se prueban diferentes combinaciones de optimizadores, con diversas ratios de aprendizaje, ya que, no se ha determinado un valor añadido por cambiar la función de activación de la red, puesto que la variación de precisión no confirma un aumento significativo.

3.4.2. Keras CNN

En cuanto al desarrollo de la red convolucional, se realizarán las mismas pruebas descritas anteriormente para determinar la mejor red que se puede obtener con los datos actuales, pero adaptando las necesidades que una red convolucional necesita.

Aspectos importantes a tener en cuenta como ya se ha comentado son: el número de capas convolucionales requeridas, el tamaño del kernel empleado para la convolución, el número de capas ocultas que se usen y las estrategias de maxpooling y regularización que se han comentado en apartados anteriores.

Primeramente, se comprobará el funcionamiento de la red sin añadir ninguna capa de convolución, ejemplo similar al clasificador MLP. Progresivamente se añadirán capas internas de convolución para comprobar si su uso está justificado para este tipo de problemas, y por último el empleo de estrategias de regularización para impedir un comportamiento no deseado de sobreajuste de red.

3.4.3. Serialización de modelos de prueba

Una vez decidido el modelo con la mayor precisión, se guardará para su uso en el servidor Python para que la aplicación de Ionic pueda mandar un nuevo dibujo, pedir al modelo que haga una predicción, y devolver los resultados correctos al usuario.

3.5. Conexión Aplicación – Servidor

Para la conexión entre la parte de servidor con la interfaz de usuario, se ha empleado Flask. El modelo MVC permite manejar las peticiones de parte del usuario de una manera cómoda y sencilla.

Actualmente Flask implementa un servidor local de pruebas no apto para uso comercial, que permite asignar la IP del ordenador que se conecte al servidor de manera automática.

El procedimiento es muy sencillo ya que únicamente necesitamos enviar una cadena en base64 con el contenido de la imagen que se ha dibujado. El servidor la decodifica y se le aplica el mismo proceso que se ha descrito en el apartado 3.3.3.

Este envío de datos se puede hacer por método GET o POST. Se ha decidido hacer por POST, aunque si bien no hay necesidad de usar uno por encima de otro ya que no se envía información privada de momento.

El servidor devuelve un JSON con la información que ha obtenido de la predicción realizada por el modelo. Ese archivo contiene un array con las probabilidades de que sea

uno de los 80 kanjis previstos y el orden no varía, lo que permite asignar la pantalla correspondiente en función del índice de mayor probabilidad.

4. Resultados

4.1. Modelo Clasificador MLP

Pasamos ahora a comprobar todos los apartados mencionados en el punto 3.6.1. empezando por el número de capas que propicien resultados satisfactorios con la máxima eficiencia y ahorro del tiempo posible.

Se van a comparar 6 modelos, entre ellos el modelo que viene de serie, y otras 5 configuraciones para determinar el impacto que tiene sobre la precisión el incluir más o menos nodos por capa.

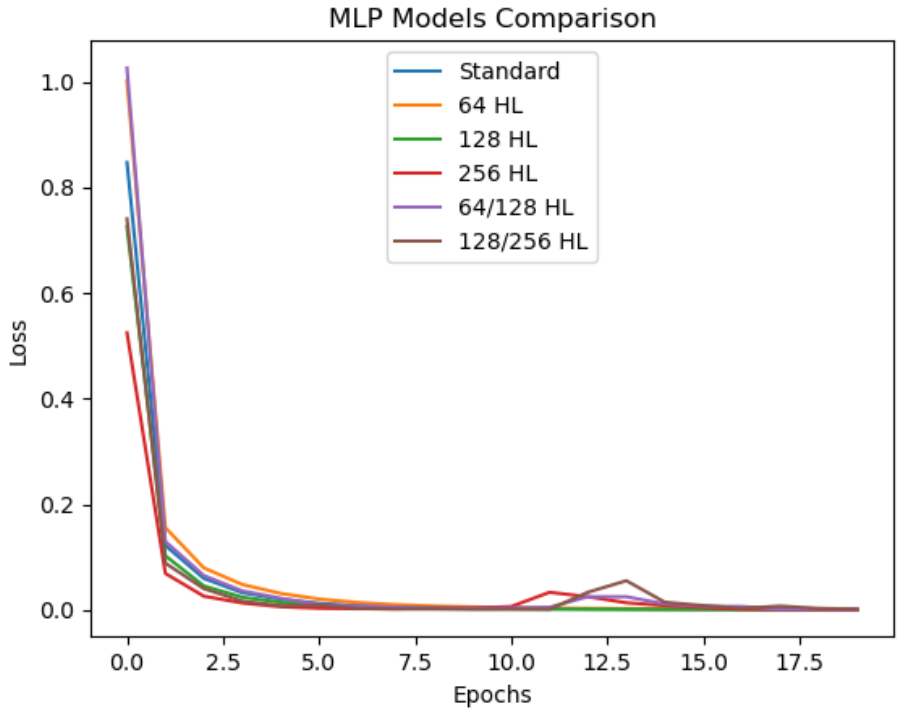

Figura 17.- Comparación entre número de capas y pérdidas por épocas

Como se puede apreciar, todas las variantes al transcurrir 20 épocas de entrenamiento, convergen en un porcentaje de pérdidas muy similar. Por supuesto eso no indica nada de los resultados de precisión de la red en sí, pero es un indicativo de que no es necesario aumentar con respecto al modelo que viene por defecto, el número de capas o la cantidad de nodos para este problema en cuestión. Obsérvese que la red con una capa oculta y 64 nodos, obtiene también unas pérdidas similares que el modelo con dos capas ocultas de 128 y 256 nodos respectivamente.

Comprobemos la media de precisión de cada modelo independiente entrenado con 30.000 muestras.

- MLPClassifier 1 – Por defecto: **96.86%** accuracy
- MLPClassifier 2 – 64 HL: **96.75%** accuracy
- MLPClassifier 3 – 128 HL: **96.97%** accuracy
- MLPClassifier 4 – 256 HL: **97.24%** accuracy
- MLPClassifier 5 – 64/128 HL: **96.92%** accuracy
- MLPClassifier 6 – 128/256 HL: **97.21%** accuracy

Se observa que nos movemos en un rango de entre 96% – 98% de precisión con 80 clases diferentes. Es un porcentaje muy satisfactorio y alto que nos permite comprobar, que la influencia de un par de capas más o unos nodos de menos no provoca una diferencia en la precisión considerable.

En subsecuentes pruebas de porcentaje de precisión, los valores han fluctuado hasta quedar por debajo de 97% aunque muy cercanos, por lo que se concluye seguir con el modelo por defecto.

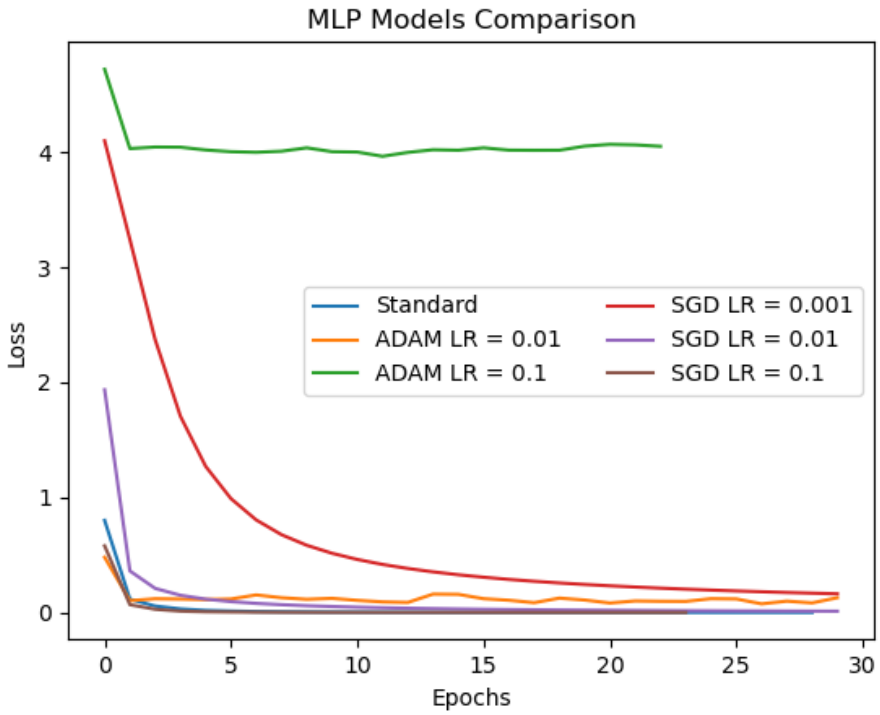

Figura 18.- Comparación entre optimizador ADAM y SGD

Continuando ahora con la comparativa de la red MLP con 100 nodos en una única capa oculta, entre el optimizador Adam y SGD. Podemos concluir en vistas de la figura 17 que, habiendo entrenado durante un periodo de tiempo más largo en vez de solo 30 épocas, todos los optimizadores hubieran convergido en un mismo valor. Queda patente que el optimizador Adam con una ratio de aprendizaje muy rápida, difícilmente aporta resultados congruentes, y se demuestra en la poca precisión que se obtiene tras el entrenamiento, 8.41%.

Es interesante destacar que el ritmo en el que entrena el optimizador SGD es más pausado y con mejor precisión en ratios de aprendizaje pequeños, sin embargo, al aumentar el valor toma similitudes con Adam y prácticamente los mismos valores. Cabe especificar que el modelo estándar utiliza Adam con una ratio de 0.001.

Comparando la predicción de modelos SGD y Adam, obtenemos un rango de precisión de entre 93% – 97%. Es un rango de un 5%, lo cual es considerable, pero en los resultados se muestra que la red aún no había convergido en los casos de SGD por lo que este porcentaje bajo podría subir con más iteraciones.

4.2. Modelo Keras CNN

Empezaremos este apartado comentando un ejemplo de red convolucional lo más cercano posible a un modelo MLP.

El primer modelo a tener en cuenta está definido con las siguientes capas.

flatten_1: Flatten	input:	multiple
	output:	multiple

dense_1: Dense	input:	multiple
	output:	multiple

dense_2: Dense	input:	multiple
	output:	multiple

Figura 19.- Descripción de capas modelo CNN

La entrada a la primera capa dense es en esencia los píxeles de cada imagen sin previo proceso convolucional, por lo que cuenta con 64x64 entradas. A su salida, entra la capa dense número 2, con tantos inputs como clases haya. En este caso 80 kanjis.

A continuación, se muestran las gráficas de precisión y pérdidas en comparativa con la fase de entrenamiento y validación.

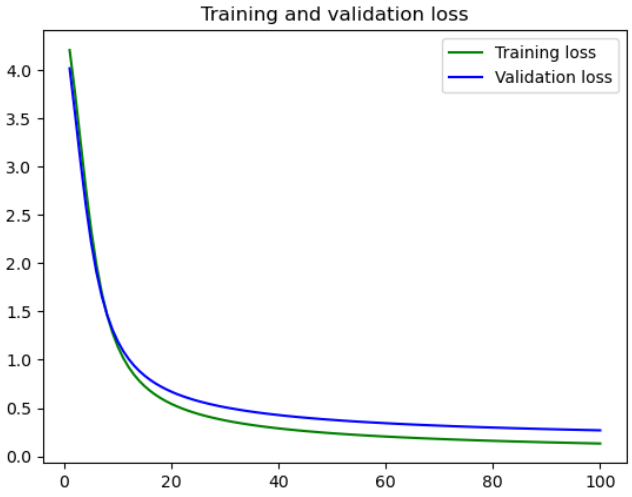

Figura 20.- Pérdidas de entrenamiento y validación CNN

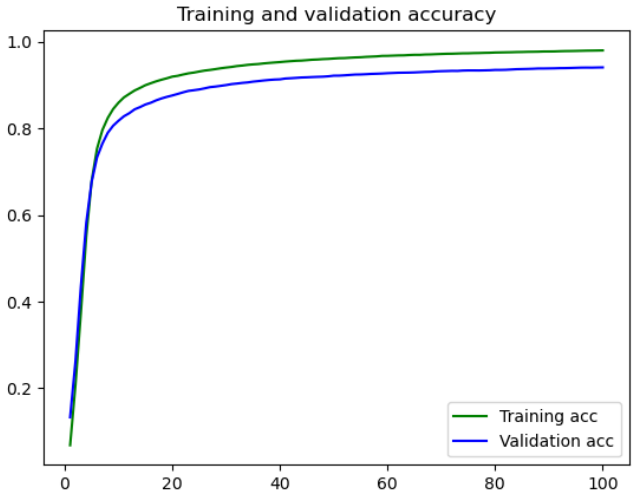

Figura 21.- Precisión de entrenamiento y validación

Comentando la figura 19, se observa como la curva de pérdidas durante el entrenamiento es superior a la de validación. Esto es normal ya que el set de datos de entrenamiento es tres veces mayor al de validación. Lo mismo ocurre en la gráfica de la figura 20, que vemos como la fase de entrenamiento generalmente obtiene mejores resultados que la parte de validación.

La separación entre curvas es un factor a tener en cuenta debido a que cuanto más separadas están entre sí, y en especial cuando la curva de entrenamiento es superior con diferencia a la de validación, se puede concluir que se sufre de overfit o sobreajuste en la red convolucional. En este caso la precisión en el set de entrenamiento está en 98,03% y el porcentaje de validación con 94,34%. Esto supone un 4% por debajo de la curva de pérdidas por lo que se debe intentar disminuir la distancia lo más posible.

En las próximas variaciones se incluyen nuevas capas para intentar mitigar este sobreajuste que sufre una red muy sencilla como la de la figura 18.

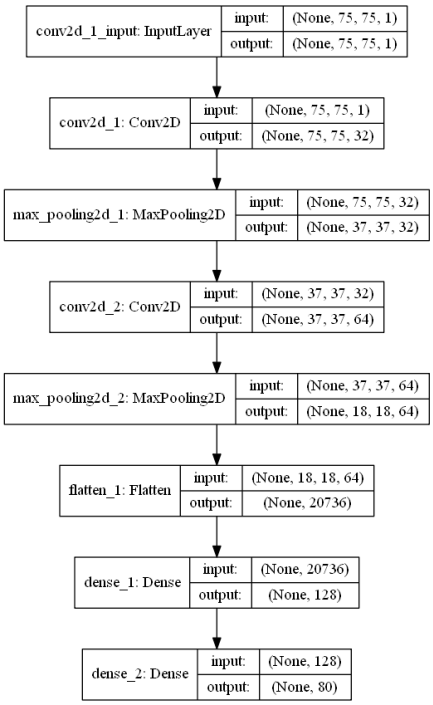

Figura 22.- Segundo modelo con capas de convolución y maxpooling añadidas

La figura 21 cuenta con la misma parte que el modelo anterior, pero con dos capas de convolución añadidas más dos capas maxpooling a la salida de ambas convoluciones. Como se ha descrito, el maxpooling resalta las características de una imagen al eliminar los valores que contribuyen poco quedando únicamente el máximo de un kernel 2x2 en este caso.

Los resultados se muestran a continuación:

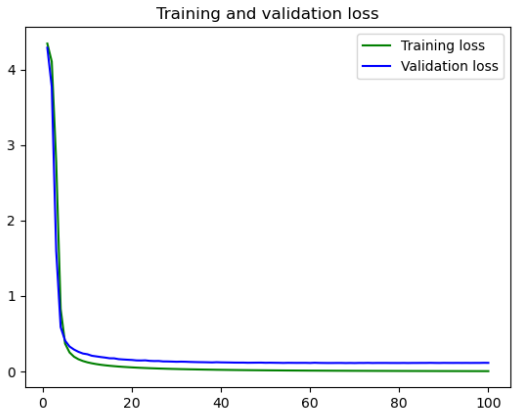

Figura 23.- Pérdidas de entrenamiento vs validación

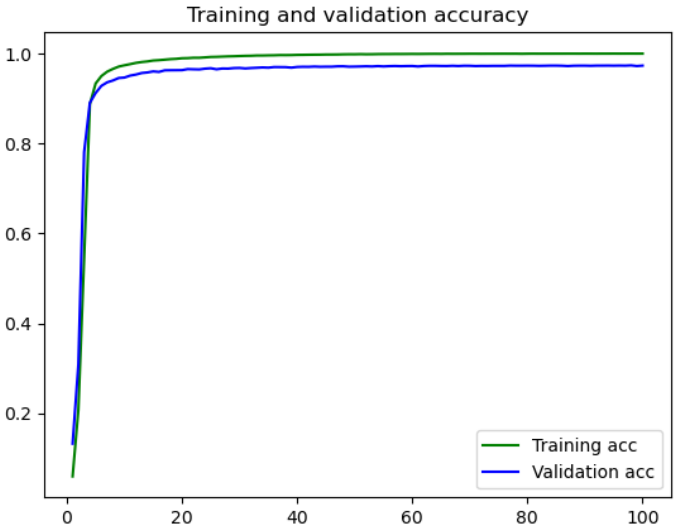

Figura 24.- Precisión de entrenamiento vs validación

Inmediatamente comprobamos que la distancia entre ambas curvas ha disminuido con respecto al anterior modelo. Sin embargo, tenemos unos valores de precisión en la fase de entrenamiento que dan a sospechar que puede haber sobreajuste en la red ya que se obtienen resultados cercanos al 100% de precisión. Es un efecto no deseado porque la red parece saber resolver este set en particular de una manera formidable, pero ante datos nunca vistos, la red presenta una disminución de un 3% con respecto a la fase de entrenamiento, dando valores de 97,48%.

Cabe resaltar que estos resultados son realmente buenos en comparación con el modelo anterior que rondaba entre un 94 y 95 % habiendo conseguido un aumento de un 3%.

Para evitar problemas de sobreajuste de la red, se empleará, en este último modelo a comparar, los términos de regulación ya comentados, en este caso las capas de *Dropout*.

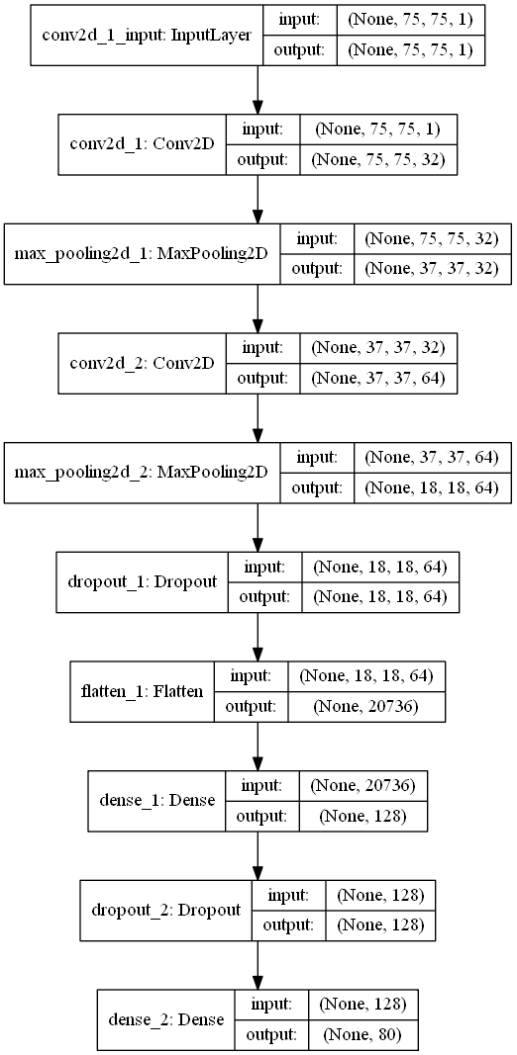

Figura 25.- Tercer modelo a comparar con términos de regularización añadidos

Para concluir esta sección, comprobaremos la importancia de añadir estos términos de regularización como buenas prácticas a la hora de evitar posibles problemas de sobreajuste.

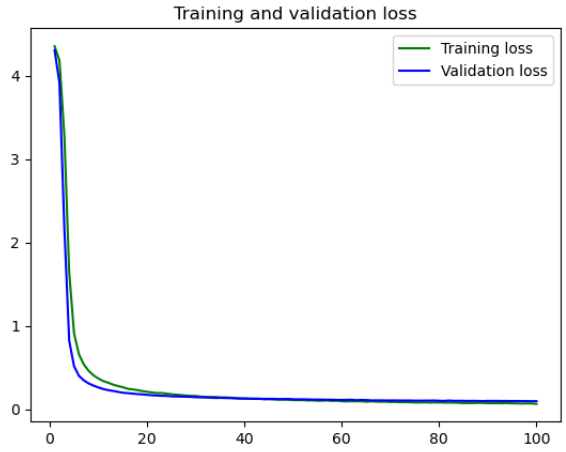

Figura 26.- Pérdidas en la fase de entrenamiento y validación

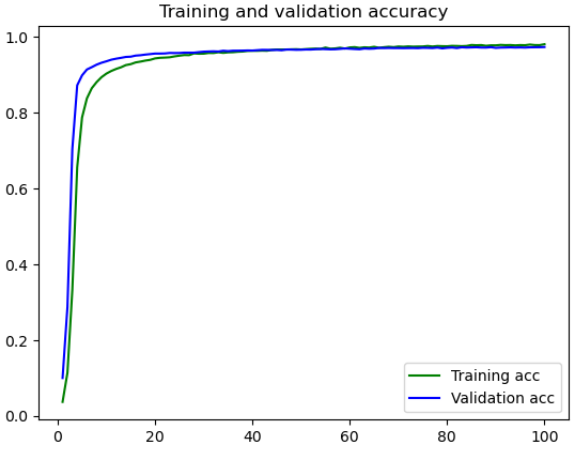

Figura 27.- Precisión en la fase de entrenamiento y validación

Finalizamos con unas gráficas que demuestran que el uso de los términos de regularización son un buen método de control de la fase de entrenamiento, consiguiendo una diferencia mínima entre la curva de validación y la de entrenamiento, y con unos valores de precisión de 97,66%.

Dado que este modelo obtiene casi un 98% de precisión y teniendo en cuenta el modelo MLP conseguía apenas llegar a un 97%, se decide emplear el modelo de convolución para su uso en la aplicación puesto que a la hora de escalar el problema y añadir más clases para predecir, los modelos convolucionales claramente obtienen ventaja en cuanto a personalización y adaptación a nuevos problemas cada vez más complejos.

5. Conclusión

Concluir este trabajo con un grado de satisfacción positivo en vistas a los resultados obtenidos tanto de la red convolucional como el clasificador multicapa y es que, con cerca de un 98% de precisión para la red convolucional, es posible predecir kanjis en la aplicación con un grado consistente de aciertos.

Comparando estos resultados con algunos de los trabajos similares en el ámbito de reconocimiento de caracteres, el trabajo citado en el apartado 2.5: **Aplicación de Deep Learning a la Enseñanza de la Escritura Japonesa**, incluye una red neuronal para reconocer en este caso los alfabetos de hiragana y katakana, con un total de 92 caracteres diferentes, y mostrando unos sólidos resultados de 99% en ambos alfabetos. Este proyecto ha demostrado no quedarse atrás a la hora de reconocer figuras más complejas como son los kanjis.

Uno de los objetivos primordiales era no necesitar saber qué trazos realizar ni en qué orden para buscar un kanji que estás leyendo y el modelo aporta resultados muy positivos en este aspecto. Claramente la forma de dibujo influye en cómo se comporta la red y serán necesarios más datos para entrenar y mejorar aún más la precisión y generalización del modelo.

La integración del modelo en una aplicación multi plataforma desarrollada en Ionic, permite que alcancemos a un público mucho mayor a si se hubiera desarrollado en espacios de trabajo como Android Studio para aplicaciones de Android, Swift para iOS o Visual Basic .NET para Microsoft para publicar la aplicación en una plataforma en concreto.

Con una interfaz sencilla y útil, que permite al usuario realizar lo que quiere en el menor tiempo de espera posible dado que es abrir y dibujar. Se considera que la herramienta será capaz de ayudar a los usuarios a seguir mejorando en el aprendizaje del idioma japonés y permitirá a los recién iniciados perder el miedo a un tópico tan grande y complejo como son los kanjis. Queda un largo camino para completar todos los niveles de kanji exigidos de uso diario pero este comienzo es muy prometedor para continuar implementando cada vez más funcionalidad en la aplicación.

6. Bibliografía

[1] «Toptal,» [En línea]. Available: https://www.toptal.com/machine-learning/introduccion-a-la-teoria-de-aprendizaje-de-maquina-y-sus-aplicaciones-un-tutorial-visual-con-ejemplos. [Último acceso: 4 Julio 2020].

[2] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Supervised_learning#:~:text=Supervised%20learning%20is%20the%20machine,a%20set%20of%20training%20examples.. [Último acceso: 7 Julio 2020].

[3] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Unsupervised_learning. [Último acceso: 7 Julio 2020].

[4] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Aprendizaje_semisupervisado. [Último acceso: 7 Julio 2020].

[5] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Reinforcement_learning#:~:text=Reinforcement%20learning%20(RL)%20is%20an,supervised%20learning%20and%20unsupervised%20learning.. [Último acceso: 7 Julio 2020].

[6] «medium,» [En línea]. Available: https://medium.com/sifium/machine-learning-types-of-classification-9497bd4f2e14. [Último acceso: 7 Julio 2020].

[7] D. Calvo, «diegocalvo.es,» [En línea]. Available: https://www.diegocalvo.es/perceptron/. [Último acceso: 7 Julio 2020].

[8] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Perceptr%C3%B3n_multicapa. [Último acceso: 7 Julio 2020].

[9] «Machine learning mastery,» [En línea]. Available: https://machinelearningmastery.com/neural-networks-crash-course/. [Último acceso: 7 Julio 2020].

[10] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Gradient_descent. [Último acceso: 7 Julio 2020].

[11] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Propagaci%C3%B3n_hacia_atr%C3%A1s#:~:text=La%20propagaci%C3%B3n%20hacia%20atr%C3%A1s%20de,propagaci%C3%B3n%20%E2%80%93%20adaptaci%C3%B3n%20de%20dos%20fases.. [Último acceso: 7 Julio 2020].

[12] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales. [Último acceso: 7 Julio 2020].

[13] «Towards data science,» [En línea]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Último acceso: 7 Julio 2020].

[14] J. Brownlee, «Machine learning mastery,» [En línea]. Available: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/. [Último acceso: 7 Julio 2020].

[15] «Papers,» [En línea]. Available: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks. [Último acceso: 7 Julio 2020].

[16] «Sitio Big Data,» [En línea]. Available: https://sitiobigdata.com/2019/05/01/imagenet-reconocimiento-visual-a-gran-escala/#. [Último acceso: 7 Julio 2020].

[17] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Funci%C3%B3n_SoftMax. [Último acceso: 7 Julio 2020].

[18] «datasmart,» [En línea]. Available: https://datasmarts.net/es/que-es-un-optimizador-y-para-que-se-usa-en-deep-learning/. [Último acceso: 7 Julio 2020].

[19] «jmlr,» [En línea]. Available: http://jmlr.org/papers/v15/srivastava14a.html. [Último acceso: 4 Julio 2020].

[20] S. B. Soumendu Das, «Researchgate,» [En línea]. Available: https://www.researchgate.net/publication/273311050_An_Algorithm_for_Japanese_Character_Recognition. [Último acceso: 7 Julio 2020].

[21] J. P. Małgorzata Gr˛ebowiec, «Researchgate,» [En línea]. Available: https://www.researchgate.net/publication/327893142_A_neural_framework_for_online_recognition_of_handwritten_Kanji_characters. [Último acceso: 7 Julio 2020].

[22] G. G. L´opez, «uc3m.es,» [En línea]. Available: https://e-archivo.uc3m.es/handle/10016/29902. [Último acceso: 7 Julio 2020].

[23] N. Elenkov, «Google Play,» [En línea]. Available: https://play.google.com/store/apps/details?id=org.nick.kanjirecognizer. [Último acceso: 7 Julio 2020].

[24] C. Colburn, «Kanjistudyapp,» [En línea]. Available: http://kanjistudyapp.com/. [Último acceso: 7 Julio 2020].

[25] A. S. Innes, «Google Play,» [En línea]. Available: https://play.google.com/store/apps/details?id=com.asji.kanjitree. [Último acceso: 7 Julio 2020].

[26] «Ionic Framework,» [En línea]. Available: https://ionicframework.com/. [Último acceso: 7 Julio 2020].

[27] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Angular_(framework). [Último acceso: 7 Julio 2020].

[28] Nvidia, «Developer Nvidia,» [En línea]. Available: https://developer.nvidia.com/cuda-toolkit. [Último acceso: 4 Julio 2020].

[29] «Deeplearning.net,» [En línea]. Available: http://deeplearning.net/software/theano/. [Último acceso: 7 Julio 2020].

[30] «Tensorflow,» [En línea]. Available: https://www.tensorflow.org/. [Último acceso: 4 Julio 2020].

[31] «Keras,» [En línea]. Available: https://keras.io/. [Último acceso: 4 Julio 2020].

[32] «The Pallets Project,» [En línea]. Available: https://palletsprojects.com/p/flask/. [Último acceso: 4 Julio 2020].

[33] «Wikipedia,» [En línea]. Available: https://es.m.wikipedia.org/wiki/Ky%C5%8Diku_kanji. [Último acceso: 4 Julio 2020].

[34] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/M%C3%A9todo_del_valor_umbral#M%C3%A9todo_de_Otsu. [Último acceso: 7 Julio 2020].

[35] «OpenCV,» [En línea]. Available: https://docs.opencv.org/master/index.html. [Último acceso: 7 Julio 2020].

[36] «Medium,» [En línea]. Available: https://medium.com/@muskulpesent/create-numpy-array-of-images-fecb4e514c4b. [Último acceso: 4 Julio 2020].

[37] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Multilayer_perceptron#cite_note-1. [Último acceso: 7 Julio 2020].